MetaBuilder: The Diagrammer's Diagrammer.

R.I. Ferguson and A. Hunter

University of Sunderland, School of Computing, Engineering and Technology, Informatics Centre, St. Peter's Campus, Sunderland, SR6 0DD, UK. Tel. +44 (0)191-515-2754, Fax. +44 (0)191-515-2781, Email. {ian.ferguson, andrew.hunter}@sunderland.ac.uk

Keywords: *MetaCASE*

Abstract

A software tool named MetaBuilder is described. MetaBuilder's purpose is to enable the rapid creation of computerised diagram editing tools for structured diagrammatic notations. At its heart is an object-oriented, graphical meta-modelling technique - a diagrammatic notation for describing other diagrammatic notations.

The notation is based upon the concept of a mathematical graph consisting of nodes and edges. Construction of a "target tool" proceeds by drawing a meta-model of the target notation. Items in the target notation are modelled as "classes" and the syntax of the target notation such as connectivity between elements are expressed as "relationships" between the classes. The actual appearance of symbols in the target notation can be entered using a built in graphical editor. Classes can have "functions" associated with them which allows different computational behaviour to be assigned. Typically this is used to allow textual reports to be generated from a diagram.

Once the meta-model is complete, a new tool can be generated automatically. Thus the time to develop such notation specific drawing tools can be dramatically reduced. As the design of a piece of software can be expressed diagrammatically, the MetaBuilder software can be used to build itself!

MetaBuilder has its origins in the field of software engineering where it is used to generate diagrammatic notation editors for CASE tools.In such an application, automated reasoning based on the semantics of the target notation can be used to provide automated aid to the process of diagram construction.

1 Introduction

The need to express information diagrammatically is common to many disciplines from project management to electrical design. The use of software to support this activity can greatly ease the task of creating and maintaining diagrams, particularly when the diagrams are of a highly structured nature, corresponding to formal rules. The development of such software is however a complex, time-consuming and therefore expensive task.

One possible solution to this problem is to automate some or all of the process of creating such software. In order to do this, some means of describing diagrammatic notations in a machine readable form must be employed. One possible approach is to use a "meta-diagramming" notation: a diagram to describe other diagrams. A system, known as MetaBuilder and its associated notation have been developed at the University of Sunderland and used to build several structured diagram editing tools.

2 Problem - The Construction of Diagramming Software is Complex

The production of structured diagramming software is a complex and therefore time-consuming task [1]. It thus follows that it is expensive to develop a tool to support a new notation and the rapid provision of support for ad-hoc or experimental notations is impossible.

One domain where this situation is of profound significance is the area of software engineering. Structured diagrammers (known as CASE - Computer Aided Software Engineering tools or, more specifically, upperCASE tools) to support various notations used during software development projects have existed since the mid 1980's. Uptake of these tools has however been notoriously poor with probably no more than 20% [2] of all software projects employing them. Chief among the reasons for this poor uptake is the fact that CASE tools compel their users to adhere to a set of rigid processes and procedures known as a method. Unfortunately many software engineers prefer (for quite valid reasons) to use ad-hoc or custom methods for each project. For reasons of cost given above, CASE tool manufacturers have been unwilling or unable to produce tools for a limited market (i.e. tools that may only be used on one project.)

Due to the difficulties in building diagramming software a classic "Catch-22" situation arises whereby tool builders will not support new (and better) methods until engineers are using them and software engineers will not adopt a new method until it is supported by CASE tools.

3 Solution - Use Software To Build Other Diagramming Software

One solution to the problem of building diagramming tools is to (partially) automate the process. The approach taken has been to examine several diagramming tools, identify the common components of such tools and to use these components to build a generic tool. This tool is then parameterised by the addition of the description of a given notation to make a tool to support that notation. This concept is illustrated in

Figure 1.

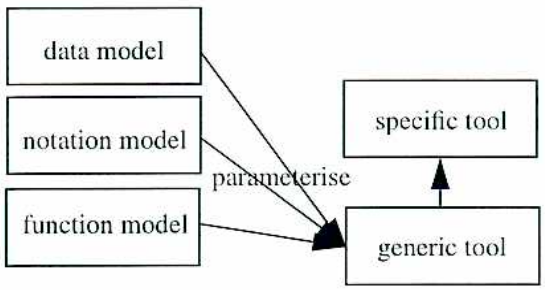

figure 1 - Generic and specific tools

When applied to the production of CASE tools, this approach is known as MetaCASE and the description of the notation as the metamodel. MetaBuilder has its origins in the field of software engineering where it is used to generate diagrammatic notation editors for CASE tools.

3.1 MetaModelling

The notation specific part of a diagramming tool is, in essence, a description of the notation that the tool will support. It specifies the things that can appear on a diagram, the relationships between them and their appearance. That description can be textual (as in systems such as TBK, MetaEdit and #others!) but an interesting possibility is the use of a diagram to provide the description.

By building a tool which supports this Metamodelling notation, and combining it with a suitable set of software components to provide the generic editing capabilities, the construction of diagramming tools can be reduced to the task of building a metamodel of a target notation and single mouse click to automatically generate the new tool. This approach can dramatically speed up the task of constructing such diagramming tools. In addition, it removes the need for skilled software engineers to construct such tools.

4 Other Related Work

Various approaches to easing the task of building diagrammers have been attempted. This sections reviews some of the most influential systems.

4.1 Other MetaCASE systems

MetaBuilder (described in section 5) is part of a "tradition" in MetaCASE which includes tools such as Ipsys TBK[14], Lincoln Software's Toolbuilder[15],

MetaEdit[16], and KOGGE[6]. The common approach to MetaCASE taken by these tools is characterised by the MetaCASE system consisting of one or more generic editors which are customised (into specific tools) by the addition of a tool description. The tool description contains a model of the underlying data repository expressed in some form of Entity-Relationship(ER) notation.

TBK/Toolbuilder TBK uses a three part textual description of tools. The underlying PCTE repository is described using Data Description Language (DDL) the graphical appearance of entities using Graphics Description Language (GDL) and the tools interface using Format Description Language (FDL). These are combined with a generic editor tool partly at compile time and partly at run time. A graphical front end to TBK known as Toolbuilder exists and obviates the need for much coding. Toolbuilder allow the creation of a metamodel using an extended ER diagram notation. Drawing tools are used to define the graphical appearance of entities.

MetaEdit MetaEdit is a Smalltalk based system which consists of an editor tool which parameterised by a tool description in the MOF language. It is not clear from documentation what MOF stands for. The tool description is similar to that of TBK is that it involves describing the entities to be edited in terms of the their attributes, appearances and the relationships in which they participate

The latest version of MetaEdit (MetaEdit+) again hides the MOF files behind graphical interface. This to relies on an extended ER notation to build the metamodel

KOGGE KOGGE's tool description is in three parts concepts (described by extended ER and GRAL - graph specification languages), menus (described by the use of Directed Acyclic Graphs - DAGs) and user interactions (specified using Harel's statechart notation). Three corresponding diagramming tools allow for the input and editing of KOGGE tool specifications.

Relationship to MetaBuilder MetaBuilder is identified as belonging to this group of tools by its use of a generic editor which is customised by the addition of a method model MetaBuilder differs from its predecessors in that it uses an entirely object-oriented approach: The generic part of the tool is provided in the form of an abstract class, the repository is described using an object modelling technique and the views of the repository are defined in terms of graphics primitive classes.

4.2. Toolkits and Component Based Approaches

Whilst the previous systems come form a MetaCASE background, MetaBuilder has similarities with systems from a diverse range of areas. One important class of system are toolkits. These systems are characterised by having no one executable program which the diagrammer builder uses but instead provide kits of software components which can be assembled to form diagram editors. In general these systems operate at a lower level than MetaBuilder, requiring (in some cases substantial) knowledge of the techniques of software development in order to produce results.

JComposer/Jviews The JComposer[10] system is a Java based system for building

multi-user graphical interfaced editors which allows simultaneous editing of objects by a group of cooperating users. Jcomposer is a tool for building tools based on the Jviews components. It provides a visual language for defining repositories, the graphical views of the repository, the user interface of the built tool and the propagation of events describing changes to items in the repository.

JComposer/JViews are based up an architecture known as change propagation and response graphs (CPRG) [11] which is designed to support consistency and versioning operations upon data in a repository. The CPRG architecture uses an ER modelling technique to define the repository, but the architecture itself is implemented as a set of OO classes. This contrasts with the MetaBuilder persistent object database which uses an OO approach both for repository design and implementation.

Xanth The Xanth system is library of software components used to support interactive structured graphics. Whilst it is certainly more advanced than a library of graphics primitives and some of the classes provided (particularly those for representing hierarchical structures and networks) are useful in constructing CASE tools, there are no data modelling or repository tools included in the system.

Escalante Escalante[18] is a system for creating structured graphics editors. It can almost be considered a MetaCASE system in that allows a model of a diagram to be expressed in entity-relationship-attribute terms but there is only limited support for building tools which do anything with the diagram once it has been drawn. Adding facilities such as report writers and code generators would be difficult with such a system.

Hotdraw Hotdraw is a collections of classes implemented in the Smalltalk language that can be assembled to produce diagram editors. No specific support is provided for representing the structure of the target notation.

Relationships to MetaBuilder The Xanth system is too low a level to be considered a MetaCASE tool. The classes that it provides are however useful to toolbuilders and would be useful should MetaBuilder be ported to a C++ environment. Escalante is in many ways similar to MetaBuilder and the capabilities of the two system intersect. The lack of access to the underlying repository however precludes its use for some task in tool building.

4.3 Visual programming languages

One of the design objectives of MetaBuilder was to build a graphically rich environment. As experience in using MetaBuilder increased, it became clear that some of the graphical notations that it was being used to develop tools for were complex enough to be considered visual programming languages.

A number of visual languages were reviewed but in the majority of cases, the environment associated with the language has been built either "from scratch" or with the aid of a graphics toolkit rather than using any dedicated automated support. A notable exception to this was the Vampire environment.

Vampire Vampire[17] is a system for developing visual programming languages. Languages built with Vampire are defined using a rule-based approach. A form of pattern matching is used to compare input in the form of "iconic" diagrams with graphically expressed rules. The pattern matcher decides whether a rule "fires" or not and the output is expressed as a diagram which has been transformed by the rule.

Elements of the language defined with Vampire become part of Vampire class hierarchy but the concept of class in Vampire is somewhat unusual in that the classes have attributes which can be either values or rules rather than data members and function members familiar from (say) C++.

Relationships to MetaBuilder Although like MetaBuilder, Vampire is based upon an OO approach to building OO language the similarity soon ends. Vampire concept of class and use of rules is significantly different to that of MetaBuilder. Vampire is aimed principally at producing executable visual languages rather than the structured diagram editors familiar with MetaCASE. It would be an interesting exercise to attempt the production of CASEtools using Vampire.

5 MetaBuilder

This section provides a description of the MetaBuilder notation and a simple example of its use.

5.1 History

MetaBuilder has its origins in a series of diagram editors created to support a software engineering method called MOOSE (Method for Object Oriented Software Engineering) [8]. These were engineered in an object-oriented (OO) fashion and as such it soon became obvious that the tools had many of their classes (or components) in common. This set of classes was gradually refined and became known as the MetaMOOSE OO component library. As MetaMOOSE [7] was used in more projects, it became apparent that much of the code needed to combine these generic components into specific tools could be generated automatically from a suitable graphical notation. That notation and its associated tool become known as MetaBuilder.

5.2 The Metabuilder Metamodelling Notation

The metamodelling technique is best introduced by example. Consider, the diagram shown in Figure 2. This represents a simple type of structured diagram (which we name box-o-method) consisting solely of labelled nodes joined by edges. In this section, the metamodel representing this kind of diagram is developed, introducing the

various modelling concepts in the process.

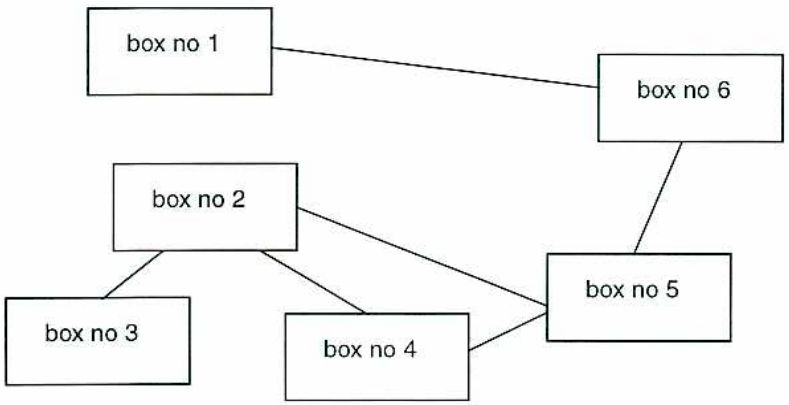

figure 2 - An example box-o-method diagram

Before describing the MetaBuilder notation, it is important to distinguish between the metamodelling notation (that used to describe the specific component of a diagrammer and the target notation (that which is being modelled in this case box-o-method).

The MetaBuilder metamodelling notation is based upon the concepts of object orientation (OO). Any OO modelling notation could have been used as the basis of MetaBuilder, but it was convenient to derive the MetaBuilder tool from an existing object diagrammer built as part of the MOOSE notation.

Classes The basic modelling concept in MetaBuilder is therefore the class. The concept is typically introduced (informally) to undergraduates by saying that any "blobs" in the target notation, must be represented by classes in the metamodel.

In the box-O-method example, there is clearly only one type of "blob" - the node - and would be modelled as shown in Figure 3

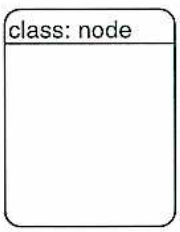

figure 3 - Metamodel of box-o-method - Stage 1

Relationships The edges in box-o-method are modelled using relationships. Informally, any lines in the target notation become relationships in the metamodel. Stage 2 of the metamodel of box-o-method is shown in Figure 4

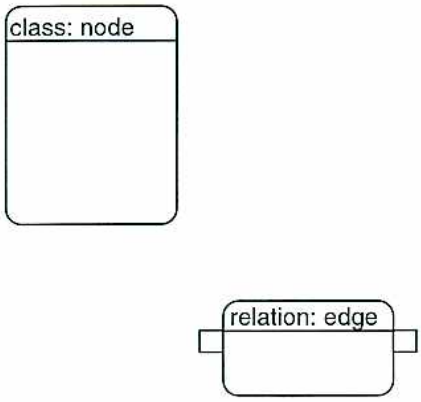

figure 4 - Metamodel of box-o-method - Stage 2

Class->Relationship and Relationship->Class Relationships It is necessary to model the manner in which the nodes and arcs may be connected to form a "legal" diagram. This is achieved by designating which type of class maybe at the source of a relationship and which at the destination. This is achieved using two types of link between classes and relationships. Figure 5 shows that nodes may be at the source of edges and shows that nodes may also be at the destination of edges.

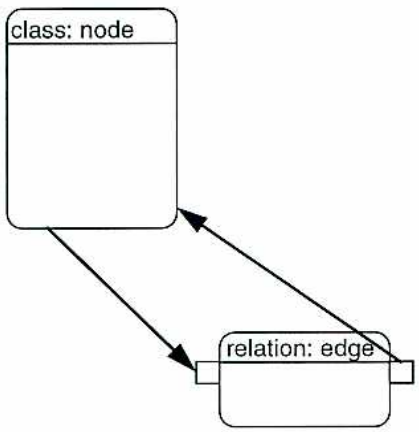

figure 5 - Metamodel of box-o-method - Stage 3

Cardinality Rules Restrictions may be placed on the number of edges that may originate or terminate at a node. The restriction is modelled as an integer value

(representing the maximum permissible number of edges) placed near the appropriate link. A value of 0 represents an unlimited number of edges. Generalising this, the number of any instances of a given relationship that a class may participate in, can be restricted by the cardinality of the class->relationship and relationship->class links.

Figure 6 shows that an unspecified number of edges may originate and terminate at any node. i.e there is no restriction

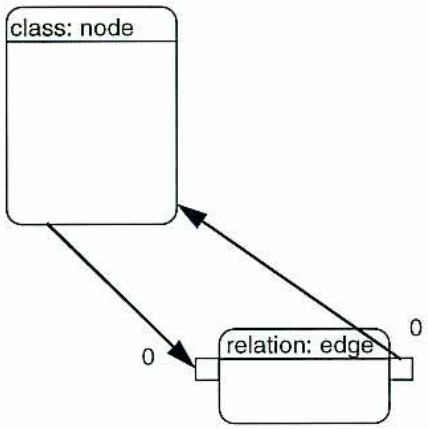

figure 6 - Metamodel of box-o-method - Stage 4

Data Members The existence of labels on nodes or edges can be modelled using data members. By adding a data member to a class or relationship, a data carrying facility is added to the underlying representation allowing the storage of textual information. Figure 7 shows the "name" data member added to the node class.

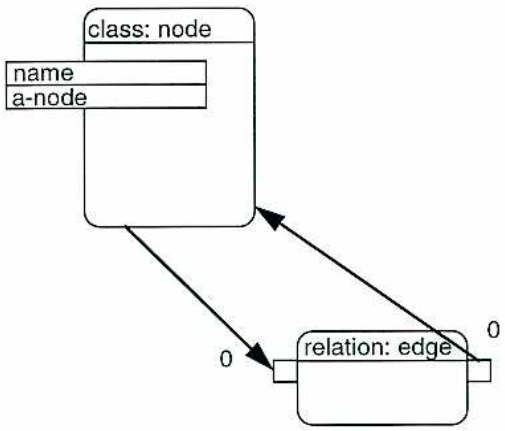

figure 7 - Metamodel of box-o-method - Stage 5

Has_a relationships A second example box-o-method diagram (Figure 8) shows that nodes may have "spots" associated with them. This can be designated in the metamodel by the use of the has_a relationship. This implies that a instance of one class maybe directly associated with an instance of another without the mediation of a line or relationship. The metamodel shown in Figure 9 demonstrates this feature.

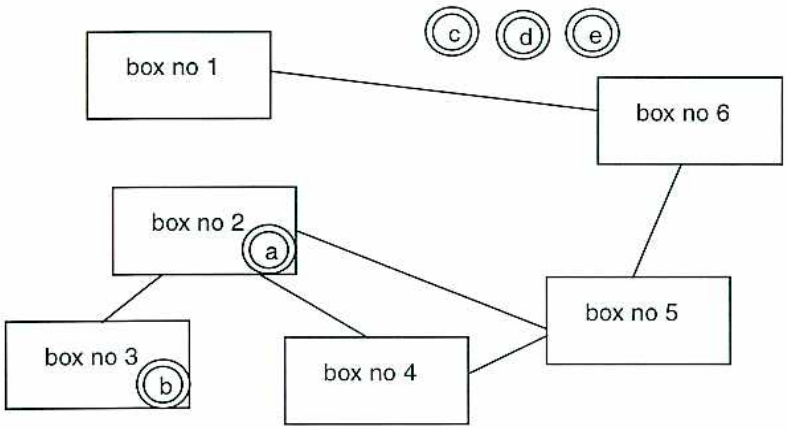

figure 8 - Example box-o-method diagram with "spots"

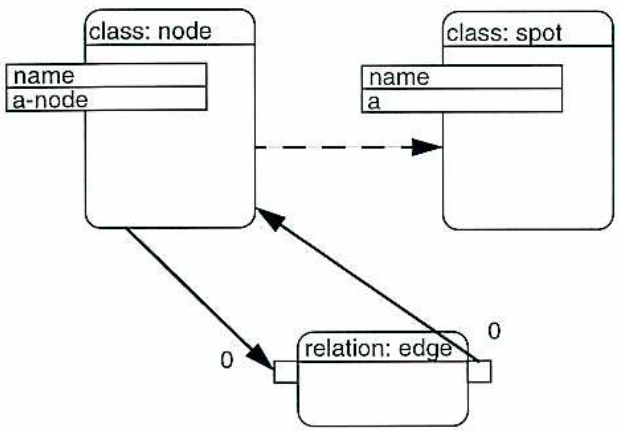

figure 9 - Metamodel of box-o-method - Stage 5

Inheritance - Variants of nodes can be created as shown in Figure 10 where two types of node are present: nodes and coded nodes which have an extra numerical label. Other than this label, coded nodes are identical to ordinary nodes, i.e they participate in the same relationships and have identical data members.

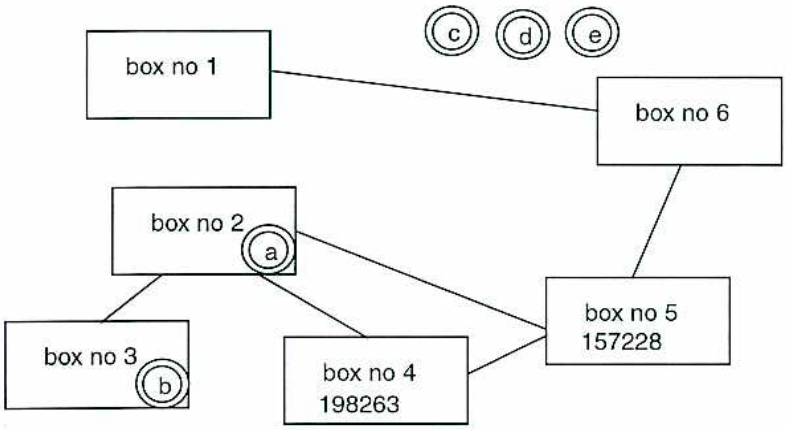

figure 10 - Example box-o-method diagram with coded nodes

In this situation, the coded node could be modelled as an entirely new class, and have the data members and relationships duplicated. This can rapidly lead to complex, hard to maintain metamodels. A much cleaner solution is to introduce the concept of inheritance (or the is_a_kind_of) relationship. The destination class of such a relationship is a special kind of the source class and inherits all its attributes. Figure 11 shows how this can be modelled. Generally, inheritance is used to create a "special

kind" of an already defined class

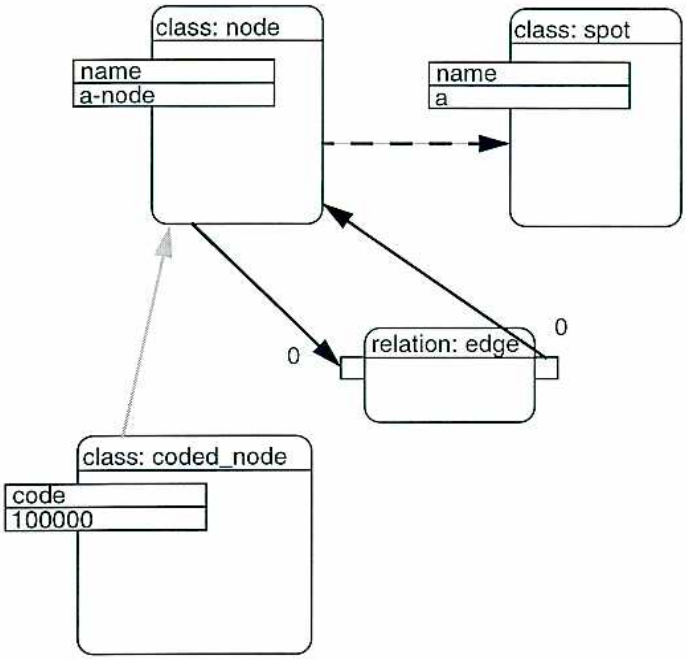

figure 11 - Completed box-o-method metamodel

Symbols The appearance of each class and relationship must be specified to complete the metamodel. The MetaBuilder system includes a simple drawing editor for creating symbols. Figure 12 shows a screen shot of the symbol editor being used to

define the box-o-method node symbol.

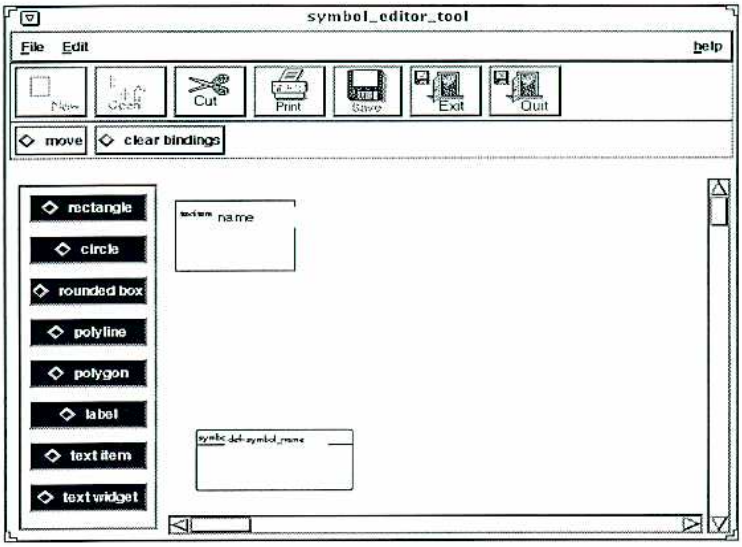

figure 12 - The Symbol Editor

Textual Annotations The symbol editor allows each data member to be associated with a user interface "widget" which allows the contents of the data member to be edited. Three types of widget are currently supported: *textitems* which allow single lines of text to edited, *textwidgets* which support multiple lines of text and *popup_editors* which allow more sophisticated editing operations to be performed. In addition, simple non-editable labels maybe added to a symbol for non-varying textual annotations.

Other notation items Various other items of notation can appear on a metamodel such as "tools" and "root objects", but these are "housekeeping" items imposed by the method of implementation. They have no bearing on the metamodelling technique and as such are beyond the scope of this paper.

A screenshot of MetaBuilder being used to prepare the metamodel of box-o-

method is shown in Figure 13

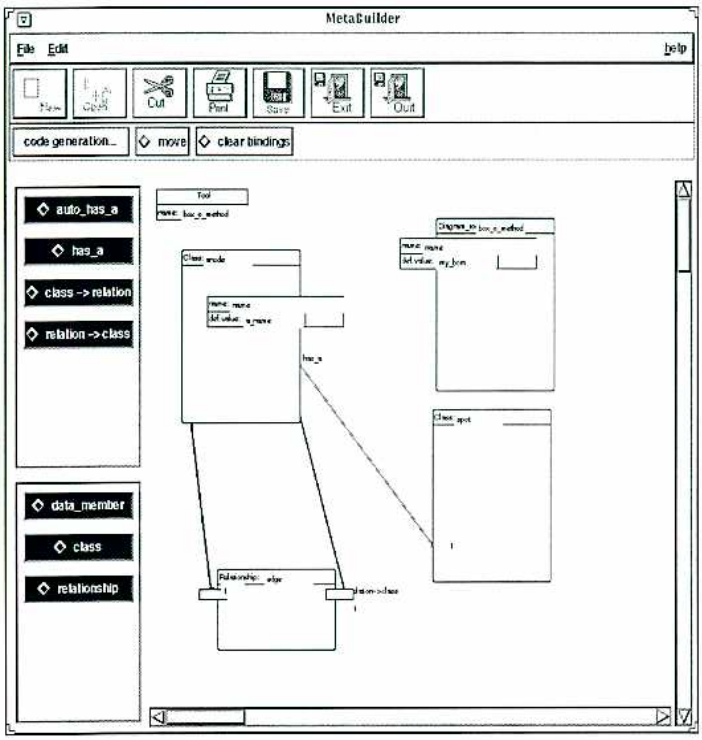

figure 13 - Screenshot of the completed metamodel

5.3 The MetaBuilder Tool

Generating a tool Once the metamodel is complete, building the target tool is simply a matter of pressing a button. Figure 14 shows the completed box-o-method

tool in use.

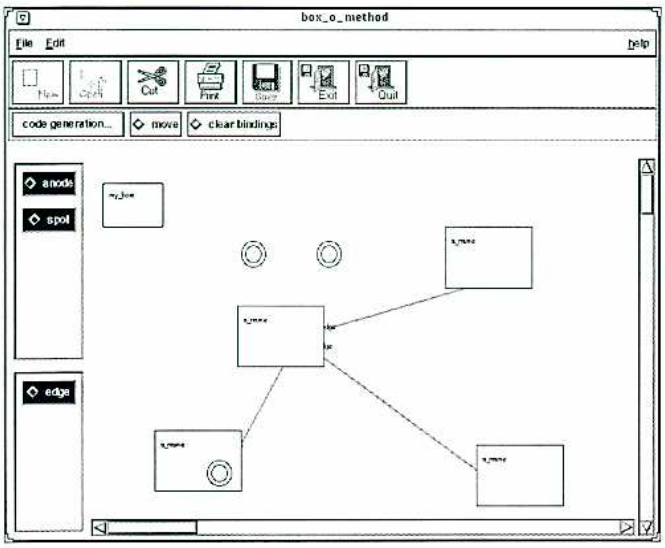

figure 14 - Screenshot of the generated box-o-method tool

5.4 Other Facilities of the MetaBuilder System

Widgets The textitems and text widgets are examples of a general class of items that can be placed on a diagram called widgets. These are items such a buttons, checkboxes and menus familiar from the graphical user interface.

Functions Allow computational behaviour to be added to classes. Triggered by user interaction with widgets or some drawing action such as mouse clicking, symbol movement or dragging and dropping these functions can access the data stored in data members. The functions are expressed in the same computer language used to implement the MetaBuilder system - itcl [20].

HyperCASE Navigation The term hyperCASE [5] refers to the ability to easily access related diagrams (or different views of the same diagram) by simple mouse operations on a diagram. MetaBuilder facilitates such navigation by the use of functions.

Multimedia A set of special widgets can be placed on a diagram which allow access to the multimedia facilities of a computer. Video widgets allow short clips of digitised video to be embedded in a diagram. Audio widgets perform the same task for digital sounds.

Report generation - code generation A common requirement having drawn a diagram (at least in the field of computing) is to have a textual report generated automatically from it. This kind of facility is the basis for code generation systems in CASE tools. A default code generation function is present in all classes in the

metamodel. This default function prints out the values of each of the class's data members and then calls the code generation function of any related classes. This default function can easily be customised to provide the desired format of generated report.

Active diagrams The facilities described in section 5.4 allow the construction of "active diagrams" - diagrams with some associated behaviour. Examples of such system are described in section 6.2

5.5 The Advantages of the MetaBuilder Approach

The MetaBuilder approach is characterised by its use of a graphical notation based on OO techniques for metamodelling mapped directly onto an OO implementation of the system. Several practical advantages accrue from using this approach to diagrammer construction that are not all present in other approaches:

Rapid Development - Complex systems of hyperlinked diagrammers can be built in minutes rather than days. The OO metamodelling approach means that notation specifications seamlessly translate into the implementations. This allows a rapid prototyping approach to be taken to tool construction. If an end user requires changes to a tool or notation, they can be effected rapidly.

Ease of Tool Construction Experience in teaching the MetaBuilder system suggests that the task of implementing such diagramming tools can be brought within reach of undergraduate computer scientists in a matter off hours. This is due to the fact that MetaBuilder abstracts all of the underlying implementational complexity allowing the toolbuilder to concentrate on the target notation.

Ease of Enhancement Some advantages of the approach described above pertain to the construction of MetaBuilder itself rather than to the building of diagrammers or to end users. The OO approach to the system allows MetaBuilder to be extended and improved more easily than would otherwise be the case. If some extra facility is required in a tool (e.g. a new type of "widget") it can be rapidly added by using MetaBuilder to add the new facility to itself, thus making it instantly available for use in all tools built with the MetaBuilder system.

HyperCASE and Multimedia The HyperCASE facility of MetaBuilder is "Internet aware" in that it allows hyperlinks to diagrams stored on another computer entirely. When combined with the multimedia capabilities, this creates a diagram publishing facility, the possibilities of which have only begun to be explored.

6 MetaBuilder in use

MetaBuilder has been used extensively within its original intended application domain of CASE tools and to a lesser extent to build diagramming tools in other areas. This section describes some of the systems built with MetaBuilder.

6.1 MOOSE toolset

Although MetaBuilder was originally derived from the MOOSE toolset (see section 5.1) MOOSE has subsequently been rebuilt using MetaBuilder. MOOSE consists of a variety of diagramming and text editing tools which make use of the full range of MetaBuilder's capabilities. Screenshots of and output from some of the MOOSE tools show the variety of supported notations.

Analysts sketchpad tool The analysts sketchpad tool was designed to allow the creating of "back of the envelope" sketches to be created in a more formal manner. It has many of the attributes of a free form drawing tool but differs in two respects: Firstly, audio and video clips maybe embedded in the diagram allowing the software engineer to capture aspects of interviews with clients or site visits to serve as an original reference point. Secondly, any of the items on the drawing can be promoted to the status of a class in the MOOSE class diagrammer via a hyperCASE link. Figure 15shows an example analysts sketchpad diagram

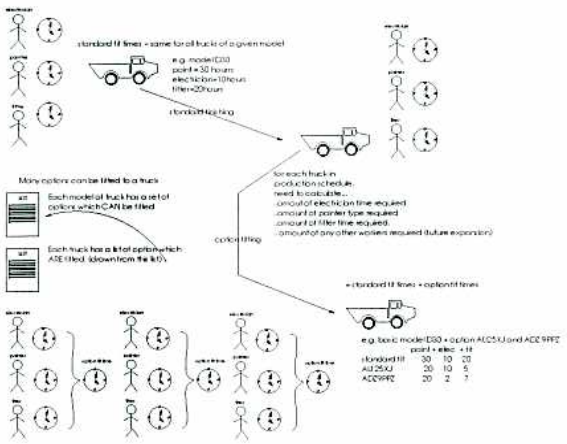

figure 15 - An example analyst's sketchpad diagram

MOOSE class diagrammer The MOOSE class diagrammer shown in Figure 16 is a typical "node and edge" diagram, supporting a variety of nodes and relationships

between them.

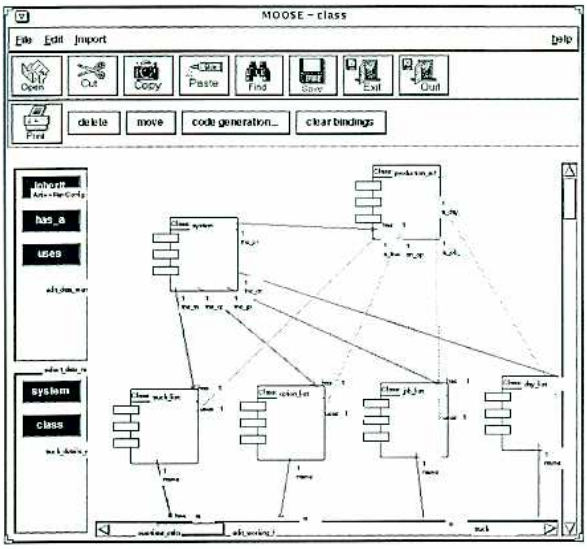

figure 16 - Class diagrammer

MOOSE object diagram The MOOSE object diagram notation (Figure 17) is composed entirely of classes with has_a relationships between them - no link relationships are present. The diagram is built up by simply dragging one node on to another. This action is sufficient to link the nodes via the appropriate has_a relationship.

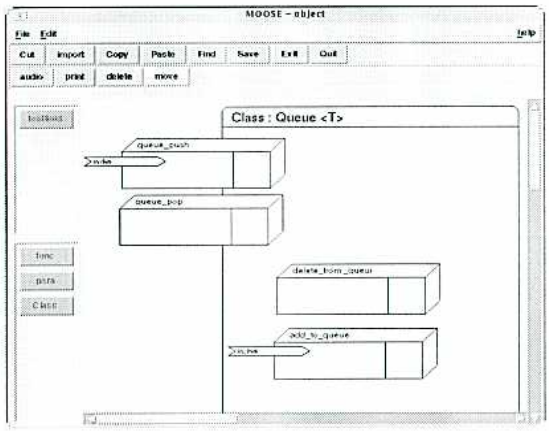

figure 17 - Object diagrammer

The VisualMOOSE tool is a user interface design tool. It allows the various

screens, forms and dialog boxes of a program's user interface to be created by dragging and dropping the components from a palette of available controls. The controls that appear on the diagram are not simply icons but are actual working controls. This means that as soon as the control is placed onto the design diagram it is fully functional and will behave as if it were part of a completed program.

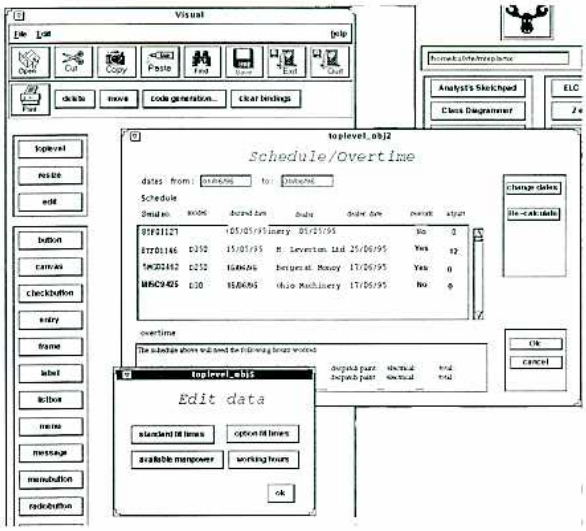

figure 18 - VisualMOOSE

6.2 Various other CASEtools

A wide range of diagramming tools has been developed for teaching purposes at the University of Sunderland, these include support for a variety of software engineering methods including Universal Modelling Language (UML), Structured Systems Analysis and Design Methodology (SSADM), Structured Systems Design (SSD) and Soft Systems Methodology (SSM). Of particular utility has been a tool to support Entity-Relationship-Attribute notation which allows database designs to be expressed diagrammatically and subsequently generates code to both create and query the database (Structured Query Language - SQL) and interface it to the World Wide Web (PHP).

Website management system bUsing diagrams to provide a graphical overview of the structure of a website and links to text editors, a website management system has been constructed using MetaBuilder.

Mobile Agent Support Tool. A tool to allow track mobile agents ([22]) as they move around the Internet has been developed with MetaBuilder [23]. It is an instance of an active diagram which automatically changes to match the current status of the system. In addition it provides a graphical means for users to interact with their agents, querying them for information and dynamically re-routing them all through a single

diagrammatic representation of the agent system.

Simulation Tool The Rezip system (#ref) is another instance of an active diagram. It is a tool which allows the building of diagrams to represent electrical power distribution systems. Once a drawing of a given network has been created, it can be animated to simulate the effect of varying loads and fault conditions.

6.3 MetaBuilder is Self-defining

Possibly the most interesting use to which MetaBuilder has been put is re-implementing itself. The MetaBuilder notation is "complete" in the sense that it can describe itself. The current version of MetaBuilder is capable of making amendments and improvements to itself. This greatly speeds up development work!

7 Conclusion

MetaBuilder has mainly been deployed to support diagrams originating in the field of computing. As such the metamodelling notation cannot be considered to be a universal notation capable of describing every other diagrammatic notation. Whilst the notation is however sufficiently general to describe most graphical notations that can be expressed in terms of the classes of object upon the diagram and the manner in which those items are related, the MetaBuilder tool itself has evolved to build a certain kind of diagrammer; i.e. CASE tools.

Within its own domain, MetaBuilder has been proven to be a versatile, rapid and powerful solution to the problems it was designed to address. Its wider applicability has been hinted at by its use in constructing tools such as Rezip and the Mobile Agent Manager. Future work on MetaBuilder will include investigating and attempting to remove the limitations of the notation as well as general improvements to the facilities offered to the diagramming tool builder.

The MetaBuilder system can be downloaded (under a shareware license) from the MetaBuilder home page [24] which also contains more complex examples of metamodelling.

References

1 Alderson A., 1991, Lecture Notes. In: *European Symposium on Software Development Environments and CASE technology, Konigswinter, Germany, June 1991* Lecture Notes in Computer Science 509. Springer-Verlag

2 Fisher, A.S., CASE: Using Software Development Tools, 2nd Ed., John Wiley & Sons, New York, NY, 1992

3 Bandinelli, S.,Di Nitto, E & Fuggetta, A., *Supporting cooperation in the SPADE-1 Environment*. IEEE Transaction on Software Engineering, Vol22, No. 12, Dec 1996

4 Campbell, I., 1986, PCTE proposal for a public common tool interface, *In:* Sommerville I., ed. *Software Engineering Environments*. Stevenage: Peter Peregrinus,

57-72.
5 Cybulski, J. L. & Reed, K., 1992. *A Hypertext BASED Software-Engineering Environment. IEEE Software*, March 1992
6 Ebert, J., Süttenbach, I.U. (1997) Meta-CASE in Practice: A Case for KOGGE, In A. Olive, J.A. Pastor: Advanced Information Systems Engineering, Proceedings of the 9th International Conference, CaiSE'97, 203-216
7 Ferguson, R.I., Parrington N.F., Dunne, P., Archibald, J.M., & Thompson, J.B., *MetaMOOSE - an Object-Oriented Framework for the construction of CASE tools*, in Journal of Information and Software Technology, Feb, 2000
8 Ferguson R.I., Parrington N.F. & Dunne P., 1994. MOOSE: A Method Designed for Ease of Maintenance. In: *Proceedings of the International Conference on Quality Software Production 1994 (ICQSP 94)*, Hong Kong: IFIP
9 Ferguson, R.I., Parrington N.F., Dunne, P., Archibald, J.M. & Thompson, J.B., MetaMOOSE - an Object-Oriented Framework for the construction of CASE tools In: Proceedings of International Symposium on Constructing Software Engineering Tools (CoSET'99) Los Angeles, May 1999
10 Grundy, J.C., Mugridge, W.B., Hosking, J.G. (1998) Visual Specification of Multi-View Visual Environments, IEEE Symposium on Visual Languages
11 Grundy,J.C., Hosking, J.G. & Mugridge, W.B., Supporting Flexible Consistency Management via Discrete Change Description Propagation, Software- Practice and Experience. Vol26, No. 9, pp 1053-1083, September 1996
12 Harmsen, F., Brinkkemper, S. (1994) Description and Manipulation of Method Fragments for the assembly of Situational Methods, Memoranda Informatica 94-52, University of Twente
13 Hill,D.H., Brinck, T., Rohall, S.L., Patterson & Wilner, W., *The Rendezvous architecture and language for constructing multiuser applications*, ACM Transaction on Computer-Human Interaction, Vol. 1, No.2 (June 1994) - pp81-125
14 IPSYS Software, 1991. *TBK Reference Manual*. Macclesfield, UK: Macclesfield, UK: IPSYS
15 IPSYS Software, 1991. *Toolbuilder Reference Manual.* Macclesfield, UK: Macclesfield, UK: IPSYS
16 Kelly, S., Lyytinen, K., Rossi, M. (1996) MetaEdit+ A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment, Lecture Notes in Computer Science, Vol. 1080, pp. 1-21
17 McIntyre, D.W. (1995) Design and Implementation with Vampire In: Burnett, M.M., Goldberg, A., Lewis, T.G. Visual Object-Oriented Programming Concepts and Environments, Prentice Hall, Chapter 7, pp.129-159
18 McWhirter, J.D. & Nutt, G.J., Escalante: An Environment for the Rapid COnstruction of Visual Language Applications, IEEE SYmposium on Visual Languages (VL'94),pp 15-22,1991
19 Ousterhout, J.K., *Tcl and the Tk Toolkit*, Addison-Wesley, Reading MA, 1994
20 Harrison, M., *Tcl/Tk Tools*, O'Reilly, 1997
21 Reiss, S.P. (1990) Connecting Tools Using Message Passing in the Field Environment, IEEE Software, pp.57-66
22 Pham,V.A & Karmouch, A., *Mobile Software Agents: An Overview*, IEEE Com-

munications Magazine, pp26-37, July, 1998
23 Grey, D.J., Ferguson, R.I. & Dunne, "WebSeeker": Efficiently locating resources on the WWW using mobile, collaborative Agents, Awaiting publication in Proc. SCS/ASIM Workshop on Agent-Based Simulation, Universitat Passau, Passau, Germany, May 2000
24 Ferguson R.I., The MetaBuilder Project, online at http://www.cet.sunderland.ac.uk/rif/metabuilder/welcome.html, Jan. 2000